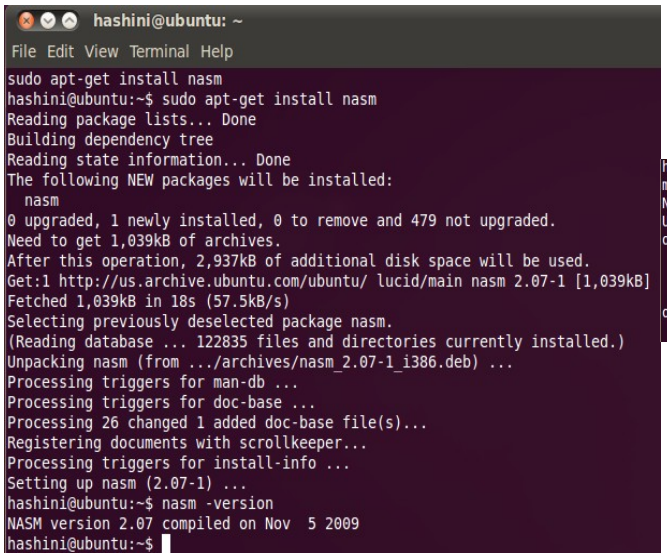
Programming Assignment : CS2042 (2010 S3)

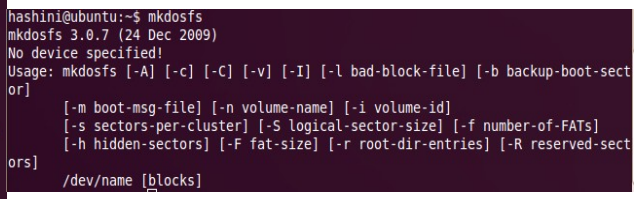
Implementing a new command for Josh operating system which prints hardware information about the computer.

Senaratne H.H.
100498G

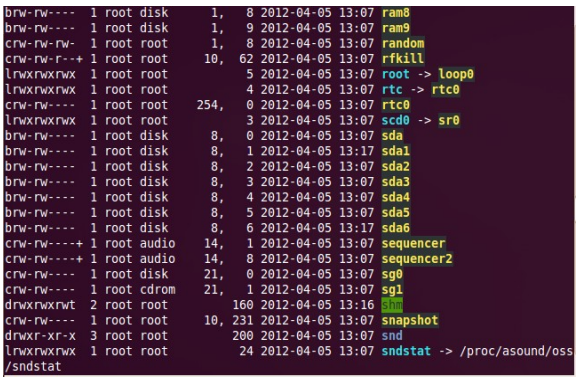
According to the guide given by the tutorial at http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/, I was able to boot the Josh operating system after overriding USB pen by the floppy image which was formatted with FAT 12. I have added some snapshots of my attempts as follows.

Setting up Netwide Assembler and DOS file utilities

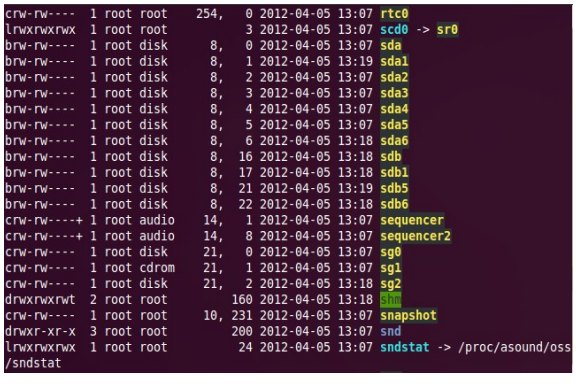




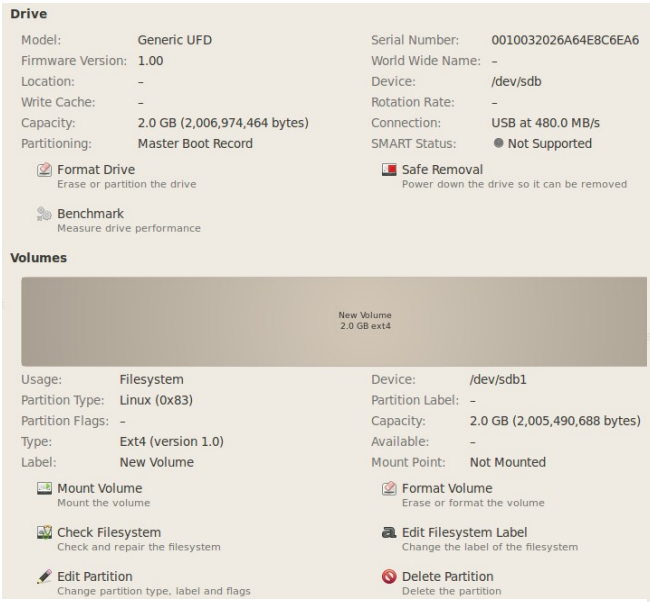
Identifying the USB device under /dev and formatting into a single disk. (in my case sdb along with the primary partition sdb1)



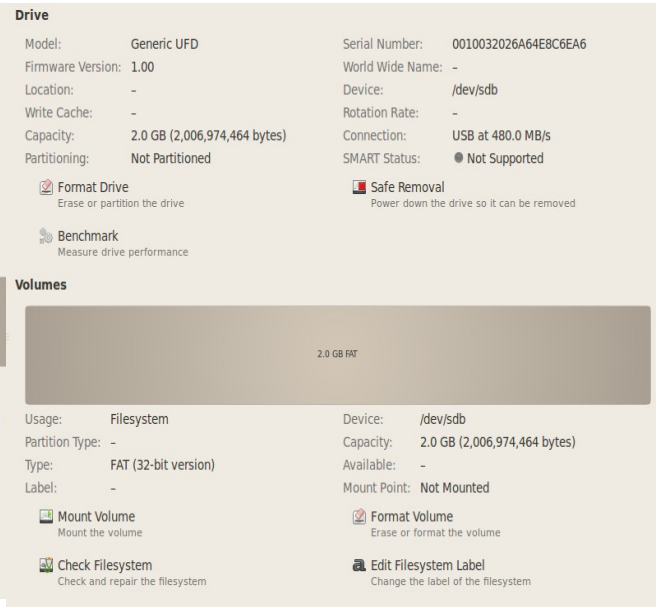
Before plug in



After plug in



Before formating



After formating

```
hashini@ubuntu:/dev$ sudo mkdosfs -F 32 -I /dev/sdb
mkdosfs 3.0.7 (24 Dec 2009)
hashini@ubuntu:/dev$
```

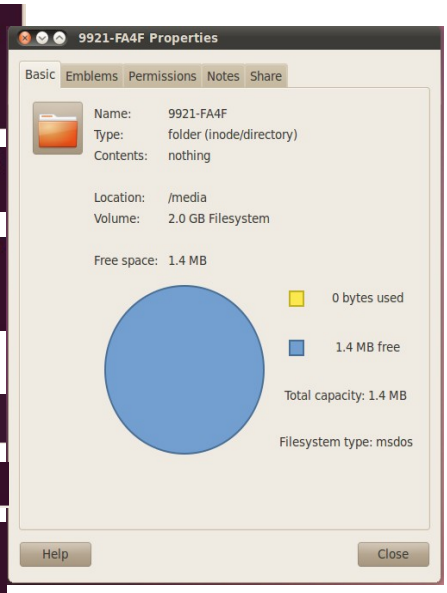
```
crw-rw-r--+ 1 root root     10,  62 2012-04-05 13:07 rfkill
lrwxrwxrwx  1 root root           5 2012-04-05 13:07 root -> loop0
lrwxrwxrwx  1 root root           4 2012-04-05 13:07 rtc -> rtc0
crw-rw----  1 root root    254,   0 2012-04-05 13:07 rtc0
lrwxrwxrwx  1 root root           3 2012-04-05 13:07 scd0 -> sr0
brw-rw----  1 root disk      8,   0 2012-04-05 13:07 sda
brw-rw----  1 root disk      8,   1 2012-04-05 13:37 sda1
brw-rw----  1 root disk      8,   2 2012-04-05 13:07 sda2
brw-rw----  1 root disk      8,   3 2012-04-05 13:07 sda3
brw-rw----  1 root disk      8,   4 2012-04-05 13:07 sda4
brw-rw----  1 root disk      8,   5 2012-04-05 13:07 sda5
brw-rw----  1 root disk      8,   6 2012-04-05 13:37 sda6
brw-rw----  1 root disk      8,  16 2012-04-05 13:37 sdb
crw-rw----+ 1 root audio    14,   1 2012-04-05 13:07 sequencer
crw-rw----+ 1 root audio    14,   8 2012-04-05 13:07 sequencer2
crw-rw----  1 root disk     21,   0 2012-04-05 13:07 sg0
crw-rw----  1 root cdrom    21,   1 2012-04-05 13:07 sg1
crw-rw----  1 root disk     21,   2 2012-04-05 13:37 sg2
drwxrwxrwt  2 root root         160 2012-04-05 13:37 shm
crw-rw----  1 root root     10, 231 2012-04-05 13:07 snapshot
drwxr-xr-x  3 root root         200 2012-04-05 13:07 snd
lrwxrwxrwx  1 root root          24 2012-04-05 13:07 sndstat -> /proc/asound/oss/
sndstat
```

Creating a floppy disk image, formatting in FAT 12 and overriding to USB.

```
hashini@ubuntu:~$ cd HackingJosh/
hashini@ubuntu:~/HackingJosh$ dd if=/dev/zero bs=512 count=2880 of=./floppy.img
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied, 0.021772 s, 67.7 MB/s
hashini@ubuntu:~/HackingJosh$
```

```
hashini@ubuntu:~/HackingJosh$ sudo /sbin/mkdosfs -F 12  ./floppy.img
mkdosfs 3.0.7 (24 Dec 2009)
hashini@ubuntu:~/HackingJosh$
```

```
hashini@ubuntu:~/HackingJosh$ sudo dd if=./floppy.img of=/dev/sdb
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied, 0.907404 s, 1.6 MB/s
hashini@ubuntu:~/HackingJosh$
```



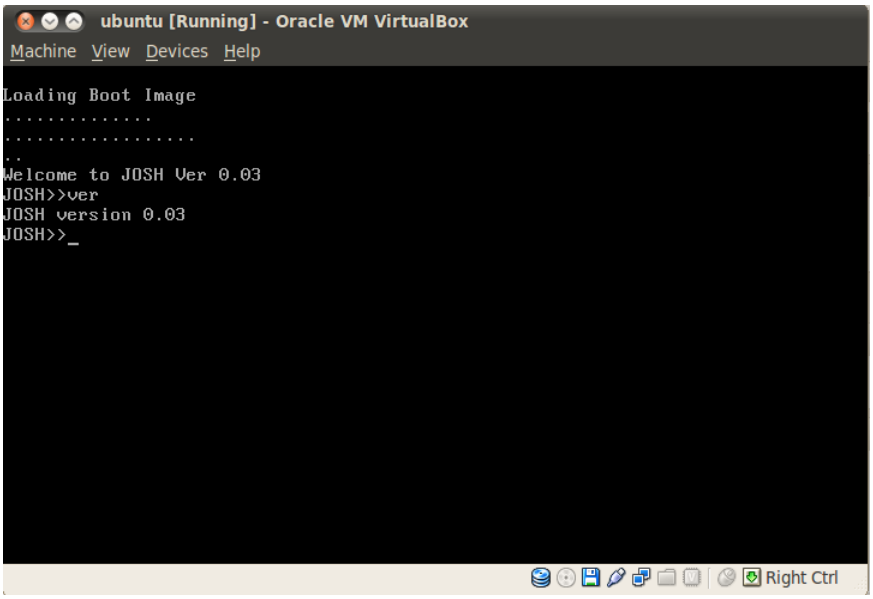
Compiling and transferring boot loader and kernel

```
hashini@ubuntu:~/HackingJosh$ nasm -f bin -o kernel.bin kernel-3.0.asm
hashini@ubuntu:~/HackingJosh$
```

```
hashini@ubuntu:~/HackingJosh$ nasm -f bin -o boot.bin boot.asm
hashini@ubuntu:~/HackingJosh$
```

```
hashini@ubuntu:~/HackingJosh$ sudo dd if=./boot.bin of=/dev/sdb
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000306952 s, 1.7 MB/s
hashini@ubuntu:~/HackingJosh$
```

For the ease of rebooting the computer frequently, I installed Oracle VM Virtual box and proceeded developing with the use of it.



Added Features

'**hw' command** which shows hardware details on the screen

- CPU vendor
- CPU type
- Mouse details
- Keyboard details
- RAM size
- Installed hard drives
- Detected Diskettes
- Detected Serial ports
- Detected Parallel Ports

'**help' command** which shows the available command list on the screen

- ver -to display version
- hw -to display hardware information
- exit -to reboot

How Achieved

First of all I had to understand the assembly codes further more and find out from which registries I can get the hardware information. For these purposes I referred the references which I have attached at the end of this document.

Hardware Information

To display hardware information I added the function "_hw" which is called through the shell using "_cmd_hw". If the user entered command matches with cmdhw string, this "_hw" function is proceeded, which calls several other sub functions to show variety of hardware details.

The source code related to this part is as follows.

```
;******************Hardware Information*************************************
_cmd_hw:                        ;;;added to shell
        mov si, strCmd0
        mov di, cmdHW
        mov cx, 1
        repe cmpsb
        jne _cmd_exit  ;next command
        call _hw
        jmp _cmd_done
_hw:                            ;;;calls to desplay hardware info
        call _display_endl
        mov si, strHardware
        mov al, 0x01
        int 0x21
        call _display_endl
        call _cpu_details
        call _mouse_details
        call _keyboard_details
        call _memory_details
        call _hardDrive_details
        call _diskette_details
        call _serial_details
        call _parallel_details
        call _bios_details
        call _display_endl
        ret

[SEGMENT .data]
        strHardware             db      "Hardware Information...",0x00
        cmdHW                   db      "hw", 0x00      ; internal commands
[SEGMENT .bss]
        strCmd0         resb    256     ;buffers for the command components
```

CPU details

```
;**************CPU details**********************
_cpu_details:
        call _display_endl      ;cpu vender
        mov si, strVendor
        mov al, 0x01
        int 0x21

        mov eax, 0x00
        cpuid
        mov [strCPUID], ebx
        mov [strCPUID+4], edx
        mov [strCPUID+8], ecx

        mov si, strCPUID
        mov al, 0x01
        int 0x21

        call _display_endl      ;cpu type
        mov si, strCpuType
        mov al, 0x01
        int 0x21

        mov eax, 0x80000002
        cpuid
        mov [strBrand],eax
        mov [strBrand+4],ebx
        mov [strBrand+8],ecx
        mov [strBrand+12],edx

        mov eax, 0x80000003
        cpuid
        mov [strBrand+16],eax
        mov [strBrand+20],ebx
        mov [strBrand+24],ecx
        mov [strBrand+28],edx

        mov eax, 0x80000004
        cpuid
        mov [strBrand+32],eax
        mov [strBrand+36],ebx
        mov [strBrand+40],ecx
        mov [strBrand+44],edx

        mov si, strBrand
        mov al, 0x01
        int 0x21

        ret
```

“cpuid” opcode is used to access the information on CPU; vendor and processor brand. Here the arguments for the opcode are given through the register “eax” and used “eax”, “ebx”, “ecx” and “edx” registers to get the results. The values in those registers are stored as a single string to get the final result.
(The **CPUID** opcode is a processor supplementary instruction (its name derived from CPU IDentification) for the x86 architecture.)

EAX=0: Get vendor ID

EAX=80000002h,80000003h,80000004h: Processor Brand String

(http://en.wikipedia.org/wiki/CPUID)

Mouse Details

```
;****************Mouse Details****************
_mouse_details:
        call _display_endl
        xor ax, ax
        int 0x11
        and ax, 0x04            ;bit 2
        shr ax, 2
        cmp ax, 0x01
        je endmouse
        mov si, strnomouse      ;if no mouse
        mov al, 0x01
        int 0x21
        jmp end1
        endmouse:               ;if detected
        mov si, strismouse
        mov al, 0x01
        int 0x21
        end1:
        ret
```

With the use of BIOS Service interrupts we can get hardware information which are stored in BIOS data area where the results are get stored in “ax” register after the interrupt call.

Here I have used the interrupt 0x11 (BIOS-Get equipment list).

Return:
(E)AX = BIOS equipment list word

Since older BIOSes do not know of the existence of EAX, the high word of EAX should be cleared before this call, if any of the high bits will be tested (xor ax,ax)

```
Bit fields for BIOS equipment list:

Bit(s)      Description

2           pointing device installed (PS)
```

Calling interrupt 0x11 will store BIOS equipment list flags in ax register. Using “and” command masks the needed bit (2nd bit). Then ax is right shifted by 2 bits to get the expected value. Then by comparing the value in ax, determine if a PS2 mouse in installed or not. Value 0x00- for no PS2 mouse detection and value 0x01- for PS2 mouse detection.

Keyboard details

```
;**************Keyboard details**************
_keyboard_details:
        call _display_endl
        xor ah,ah
        mov ah,0xf2
        int 0x16
        cmp al,0x00
        je _none
        mov si, strPC   ;if 9-bit PC keyboard
        mov al, 0x01
        int 0x21
        jmp _end2

        _none:
        cmp al,0x01
        je _AT
        mov si, strnokey ;if no keyboard detected
        mov al, 0x01
        int 0x21
        jmp _end2

        _AT:
        mov si, strAT   ;if 11-bit AT keyboard
        mov al, 0x01
        int 0x21
        jmp _end2

        _end2:
        ret
```

Here the interrupt 0x16 is used.

Int 16/AH=F2h

(Compaq 386 and newer - DETERMINE ATTACHED KEYBOARD TYPE)

Return:
AL = type
00h if 11-bit AT keyboard is in use
01h if 9-bit PC keyboard is in use
AH = 00h (04/08/93 system ROM)

By calling the above mentioned interrupt, we can gain the keyboard type of PCs which are newer than compaq 386 by looking in to the stored value at “al”.

Memory details

```
;***************memory details*********************
_memory_details:
       call _display_endl
       xor ax,ax
       xor bx,bx
       xor cx,cx
       xor dx,dx
       mov ax, 0xe801
       int 0x15
       jc _error               ; if CF is set on an error
       cmp ah, 0x86            ; check for unsupported function
       je _error
       cmp ah, 0x80            ; check for invalid command
       je _error
       mov si, strmem
       mov al, 0x01
       int 0x21
       cmp cx, 0x0000          ;if cx=0
       je _cx_zero
       jmp _mem_cal

       _cx_zero:               ;solve cx conflict
       mov cx,ax
       mov dx,bx

_mem_cal:
shr dx, 4               ;divide dx by 2^4
shr cx, 10              ;divide cx by 2^10
add cx,dx               ;get the total
mov dx, cx
call _hex2dec           ;convert hex to decimal
mov si, strMB
mov al, 0x01
int 0x21
jmp _memdone

_error:                 ;error message
mov si, strMemErr
mov al, 0x01
int 0x21

_memdone:
ret
```

Here interrupt 0x15 is used with EAX = 0xE801.

Int 15/AX=E801h (Phoenix BIOS v4.0 - GET MEMORY SIZE FOR >64M CONFIGURATIONS)

```
Return:
CF clear if successful
AX = extended memory between 1M and 16M, in K (max 3C00h = 15MB)
BX = extended memory above 16M, in 64K blocks
CX = configured memory 1M to 16M, in K
DX = configured memory above 16M, in 64K blocks
CF set on error
```

For error detection CF is checked and further more ah is compared with 0x86- to check whether the function is supported and ah is compared with 0x80- to check whether command is invalid. On some systems, the BIOS returns CX=DX=0000h; in this case, use AX and BX instead of CX and DX. This is also corrected. "_hex2dec" converts hexadecimal value to decimal. To convert into MB, "cx" is divide by 2^10 and "dx" is divided by 2^4 (dx is in 64K blocks).

Hard drive details

```
;*****************Hard drive details*****************
_hardDrive_details:
        call _display_endl
        mov si, strHardDrive
        mov al, 0x01
        int 0x21
        mov ax, 0x0040
        push es
        mov es,ax
        mov al,[es:0x0075]      ; read 40:75 offset
        add al, 48
        pop es
        mov ah, 0x0e
        int 0x10
        ret
```

Here I have accessed BIOS Data area with the use of offsets, unlike using interrupts as described under Mouse details. The BIOS data area is created at memory location 0040:0000h with a typical size of 255 bytes, when the computer powered on.

To get hard drive details we have read the offset 40:75 with the use of ax and es registers. es register value has been restored at the end with the help of stack.

Diskette details

```
;***************Diskette details******************
_diskette_details:
        call _display_endl
        mov si, strDiskette
        mov al, 0x01
        int 0x21
        xor ax, ax
        int 0x11
        and ax,0x01
        cmp ax,0x01
        je _is_floppy
        mov ah, 0x0e    ;if no floppy print 0
        mov al, '0'
        int 0x10
        ret

        _is_floppy:     ;if floppy is installed
        and ax, 0xc0    ;bit 6 and 7
        shr ax, 6
        add ax, 49      ;convert to ascii (+1)
        mov ah, 0x0e
        int 0x10
        ret
```

Just like we gain details regarding mouse, here also used the interrupt 0x11. In this case we can use 0,6 and 7 bits.

```
Bitfields for BIOS equipment list:

Bit(s)  Description
0         floppy disk(s) installed (number
specified by bits 7-6)
7-6     number of floppies installed less 1
(if bit 0 set)
```

Bit 0 is checked to see whether any floppy disk is installed. If it is set the number of floppies are read by extracting bits 6

and 7 (To convert the no of diskettes added 49 because it is less than 1, otherwise prints 0.

Serial details

```
;*************Serial details*****************
_serial_details:
        call _display_endl

        mov si, strSerial
        mov al, 0x01
        int 0x21

        xor ax, ax
        int 0x11
        and ax, 0xe00   ;bits 9-11
        shr ax, 9
        add ax, 48      ;converts to ascii
        mov ah, 0x0e
        int 0x10
        ret
```

Again used interrupt 0x11 to obtain serial details.

```
Bit fields for BIOS equipment list:
Bit(s)    Description

11-9      number of serial ports installed
```

To convert the value into ASCII, added 48.

Parallel details

```
;*************Parallel details***************
_parallel_details:
        call _display_endl

        mov si, strParallel
        mov al, 0x01
        int 0x21

        xor ax, ax
        int 0x11
        and ax, 0xc000  ;bits 14 and 15
        shr ax, 14
        add ax, 48      ;converts to ascii
        mov ah, 0x0e
        int 0x10
        ret
```

Same as above where 14 and 15 bits are extracted.

```
Bit fields for BIOS equipment list:
Bit(s)    Description

15-14     number of parallel ports installed
```

Bios details

```
;*************Bios details******************
_bios_details:
        call _display_endl
        mov si, strBios
        mov al, 0x01
        int 0x21
        push es
        mov ax, 0xf000  ;BIOS release date
        mov es, ax      ;      is in F000:FFF5
        mov si, 0xfff5
        mov bl,8

        _loop:          ;loop to print
        mov al, [es:si]
        mov ah, 0x0e
        int 0x10
        inc si
        dec bl
        cmp bl, 0
        jne _loop

        pop es
        ret
```

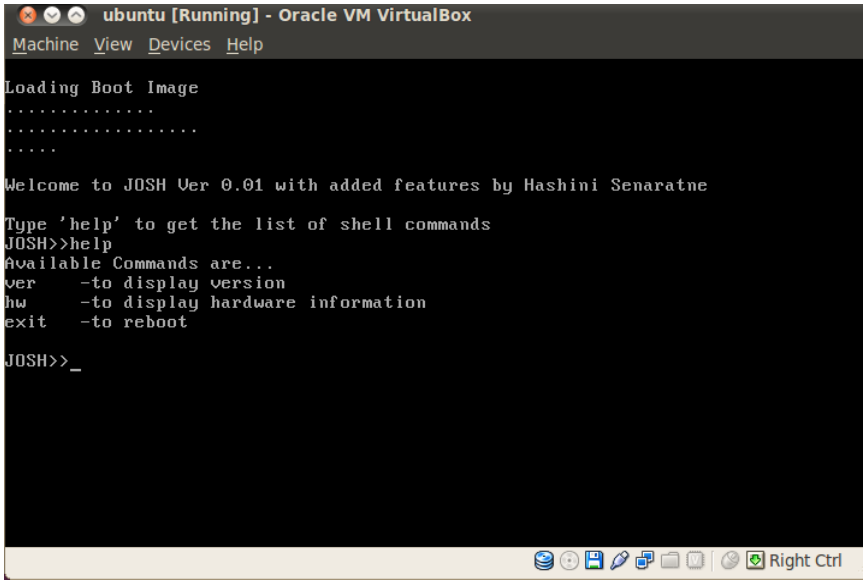
By reading the memory locations F000 to FFF5 we can obtain Bios release date in format DD/MM/YY consists of 8 characters. The loop run for 8 times which prints on character at each time.

Available Commands

```
;************Available Commands************
_help:
        call _display_endl
        mov si, strCommands
        mov al, 0x01
        int 0x21
        call _display_endl
        mov si, strVer
        mov al, 0x01
        int 0x21
        call _display_endl
        mov si, strHW
        mov al, 0x01
        int 0x21
        call _display_endl
        mov si, strExit
        mov al, 0x01
        int 0x21
        call _display_endl

        ret
```

This function prints all the available commands in the modified Josh Operating System.




```
JOSH>>ver
JOSH version 0.03

JOSH>>hw
Hardware Information...

CPU vendor                  : GenuineIntel
CPU type                    : Intel(R) Core(TM)2 Duo CPU     T5870  @ 2.00GHz
Mouse details (PS)          : Mouse is installed
Keyboard details            : 9-bit PC keyboard is in use
RAM size                    : 511MB
Installed hard drives       : 1
Detecked Diskettes          : 1
Detecked Serial Ports       : 0
Detecked Parallel Ports     : 0
Bios released               : 06/23/99

JOSH>>exit_
```

References

http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/
http://problemsolvedweb.blogspot.co.uk/2012/04/hacking-josh-with-virtual-hardware.html
http://www.bioscentral.com/misc/bda.html
http://stanislavs.org/helppc/bios_data_area.html
https://github.com/sunimalr/Hacking-Josh-OS
http://insightforfuture.blogspot.com/2012/01/hacking-josh-assembly-operating-system.html
http://www.ctyme.com/intr/int.html
http://leto.net/writing/nasm.php
http://www.posix.nl/linuxassembly/nasmdochtml/nasmdoca.html
http://www.cs.virginia.edu/~evans/cs216/guides/x86.html